Autodesk AEC DevCamp 2012

Asynchronous Interactions With Autodesk Revit API

Arnošt Löbel
Sr. Principal Engineer

Autodesk

Things can go bad in asynchronous workflows

- Why doesn’t it work?
- Where’s the danger?
- Why aren’t we more protected?
- Secured gateways to Revit
- What could possibly happen?
- Stories to learn a lesson from
- Is there a way out?

Where is the danger?

API clients make asynchronous calls mostly for good reasons, particularly to speed up Revit, but the outcome of unexpected invocations is often severe

- Documents will not be protected properly (no firewall!)
- Transaction and Regeneration mode not set
- Revit will not know the running application's ID
- The API scope is out of sync
  - therefore events and updaters may behave unexpectedly

Why aren’t we more protected?

Most of unexpected calls either cause exceptions that would not happen otherwise, or miss on exceptions that should happen. Both cases could be equally dangerous. Revit cannot handle it because:

- Revit does not have a multi-thread-ready API
- Revit uses multi-threading internally only
- Revit uses a different thread context in the API
- API Firewall is not set around each every API call

Revit doesn’t expect API calls from other than the main Revit thread!

Secured gateways

Execution tasks during which Revit expects API calls:

A. The OnStartup and OnShutdown methods
B. The Execute method of an external command
C. Execution of a VSTA macro
D. Handling an API event
E. Invoking methods of a callback interface
   - (e.g., updater, failure pre-processor, etc.)

Outside of these tasks Revit does not know your application!

Prohibited interactions

Basically, if it isn't once of the secured gateways, it is prohibited, unsupported, and dangerous:

A. Calling the API when Revit does not expect it
   - This includes subscribing to events, or adding updaters

B. Invoking Revit UI indirectly (e.g. via messages)

When the above is utilized, it is just a question of "when", not "whether" your external applications will stop working!

What could possibly happen?

A. Nothing bad at all happens
   - The call succeeds and all is fine (but would you bet on it?)

B. Calls “appear” all right and returning “successfully”
   - Though the gathered data are out of date or out of sync

C. Calls fail, exceptions are thrown
   - And this is actually a good scenario!

D. Revit crashes, document(s) must be closed
   - Frustration. Hard to determine whom to blame

E. Revit model is corrupted by two independent processes
   - In the worst case, the corrupted document is saved!

The “just-reading” case

Consider for a moment one of the less severe scenarios when the asynchronous call (AC for short) “just” reads data from the model:

1. AC calls to get one parameter of a wall
2. Another application changes the wall on Idling
3. AC calls to get another parameter of the wall
4. An updater reacts to the change by another change
5. AC calls to get another parameter of the wall

Though the calls were next to each other in the client’s code, they were technically executed at different times and the data acquired by each of the methods reflects the state of the model at those different times.

The case of crashed Revit

1. User invokes an external command
2. Command splits a worker thread and waits for it to finish
3. The thread attempts to start a transaction
4. Revit throws an unhandled exception, possibly crashes

How come this is an unsafe workflow?

- ⇨ Because Revit API was called from another thread!

What was the reason for the crash?

- ⇨ Revit allocated memory in a wrong thread's context!

Even such calls that appear as if they should be indifferent (since they look like involving the managed layer only), might not be safe, because it is likely that internal native code (and memory allocations) gets executed.

The Russian Roulette case

Calling Revit API directly from a modeless dialog is not unlike betting in Las Vegas. One can gain something sometimes, but they will get you eventually. Applying it to our case – something's going to crash.

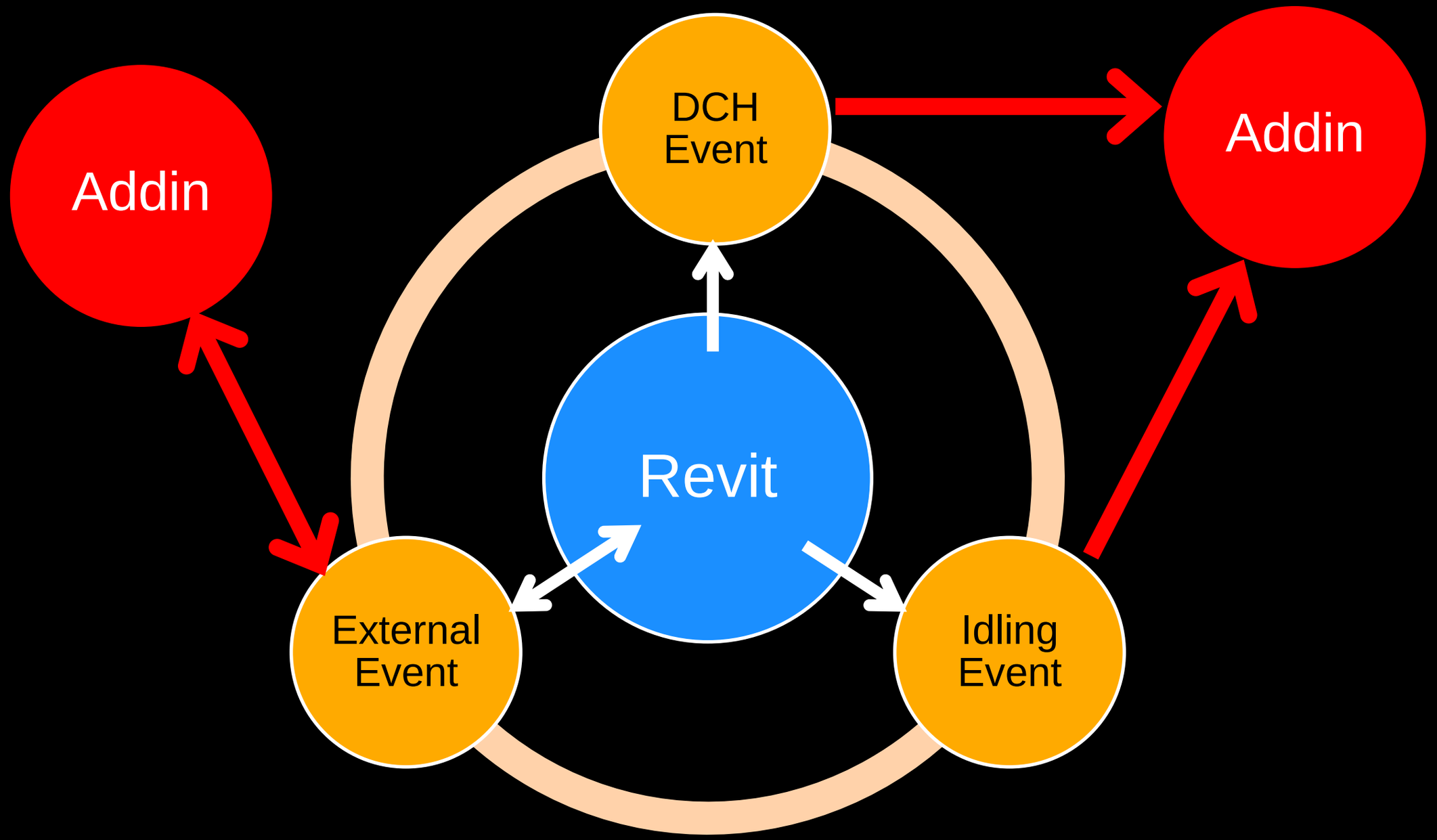
1. Revit calls an external command
2. Command launches a modeless dialog and returns
3. The dialog awaits the user's requests
4. On a request, the dialog:
   a) Uses the API to modify (or to just access) the model
   b) Sends a message to Revit's main window to invoke a command.

The outcome of such calls is virtually unpredictable.

Happy Together (asynchronously)

Autodesk

Asynchronous workflows by the book



Autodesk

Idling to the rescue

1. Use Idling to make API calls at unspecified times
2. Idling means Revit API is ready
3. DocumentChanged means Revit's done something
4. Those events are powerful when used together
5. Call to the API must be on the event's thread

The Idling event's details

- Idling is a UI-concept
  - It is in `RevitAPIUI` & the sender is `UIApplication`
  - Idling means UI is "at rest" – the external app should respect that
- Idling is raised when:
  - Revit is not doing anything and is ready to do something
  - No command executing, no editor running, no transaction open
- Idling is raised a lot, it should not be abused
  - Spend as little time in the handler as possible
  - If you need to do more, do it in another thread
  - Use idling frequency wisely (`SetRaiseWithoutDelay`)
- Just being subscribed can slow down your computer
  - Subscribe just when you need it, and unsubscribe when you don't anymore.

External Events

- A new API added in 2013
- Allows an external app to “signal” it wants to do something
  - So instead of waiting for the right Idle moment, the app specifies the time
  - It is like subscribing to idling just for that one moment
- External Events are internally still based on Revit Idling
- Solution primarily tailored for external modeless dialogs
- Some applications are still better suited for Idling Event
  - Or a combination of External Event and Idling Event

Modeless dialog pattern - Idling

Phase 1

1. An application registers itself and its commands
2. A command kicks off a modeless dialog
3. The command registers for the Idling event
4. Command returns, Revit continues running

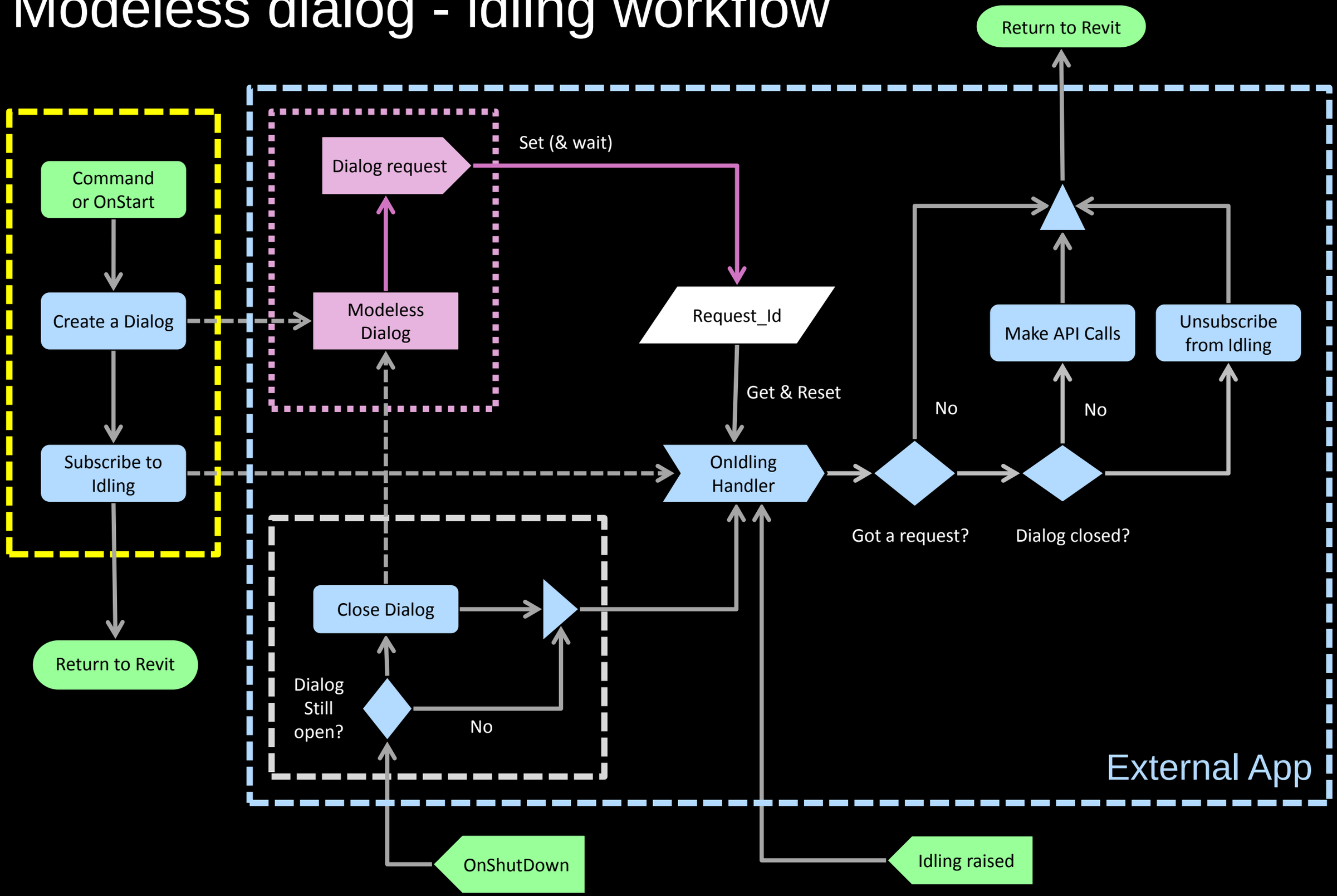
Phase 2

1. At times, Revit raises the Idling event
2. The event's handler checks if the dialog requested anything
3. If so, the handler takes the request and use the API

Phase 3

1. When the dialog is closed, it sets a flag somewhere
2. If flag is set on the next Idling, the handler unregisters itself

Modeless dialog - idling workflow



Autodesk

Modeless dialog pattern – External Event

Phase 1

1. An application registers itself and its commands
2. A command kicks off a modeless dialog and
3. The command acquires an instance of **`ExternalEvent`**
4. Command returns, Revit continues running

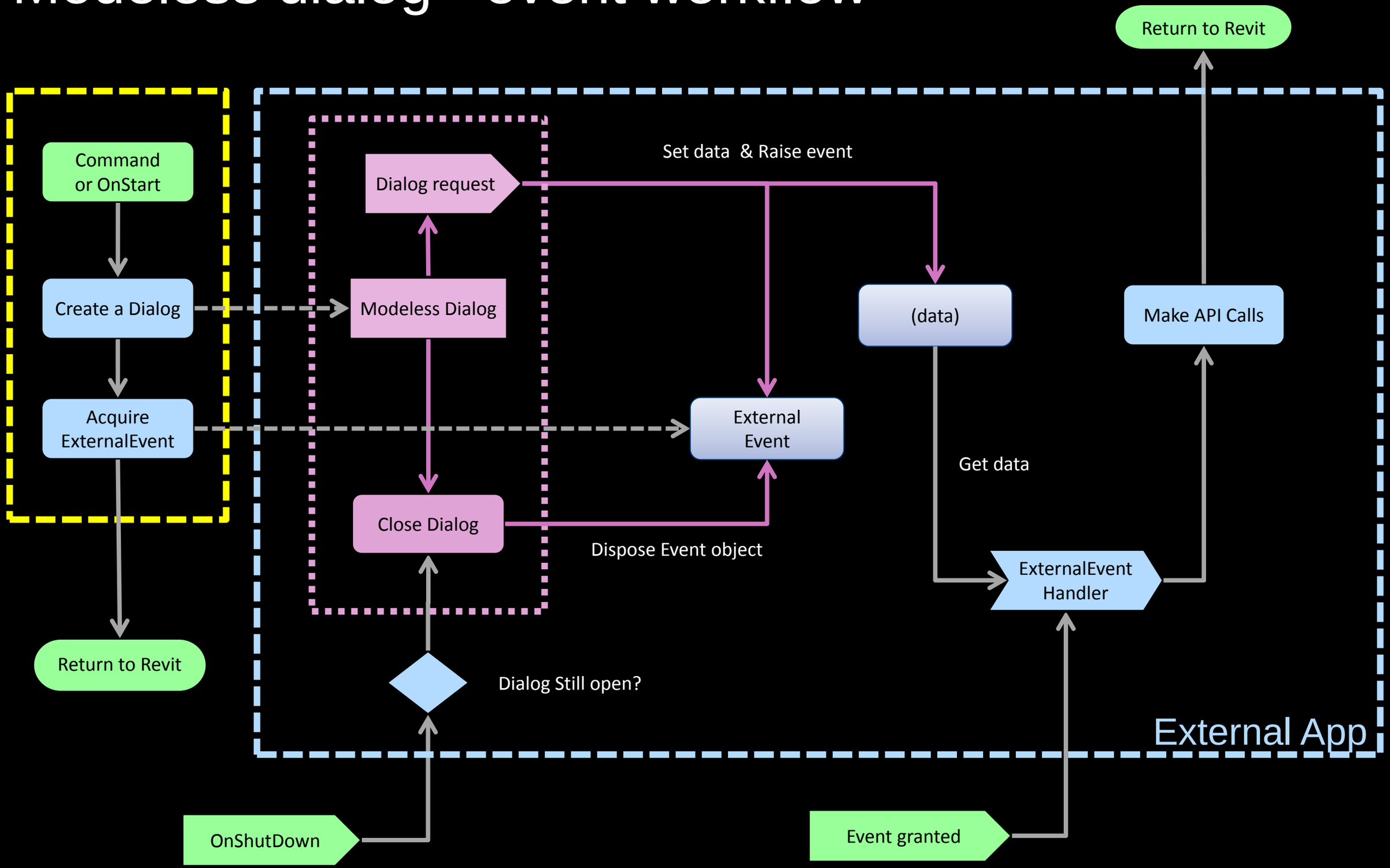
Phase 2

1. The dialog signals the event when it wants to call Revit
2. Revit waits for the right time, then calls the event's handler
3. Dialog does what it needs to in the handler's **`Execute`**

Phase 3

1. When dialog closes, it should dispose the External Event

Modeless dialog - event workflow



Autodesk

A workflow with a twist

When changes to the model on individual interaction are not atomic, the Idling event is not sufficient; the DocumentChanged event must also be utilized. The differences from the basic workflow are:

1. Subscribe to Idling and DocumentChanged
2. On each document-changed event
   - ✓ Check if any data related to the process changed, and if it did:
     1. Make sure obsolete data would be removed from the model (this would probably be done on next idling event)
     2. Restart the worker process

The DocumentChanged event

- Technically not part of a transaction
- Runs before a transaction ends but after Updaters
- It runs when a transaction is either committed or rolled back
- It also runs when transaction(s) is/are undone or redone
- Handler is not permitted to make model changes
- Handler has access to three sets of changed elements (ids):
  - Added elements
  - Modified elements
  - Deleted elements
- Primary use: synchronizing with data outside of the model

What we have learned

- ✓ I shall not call the API unless invoked by it
- ✓ I shall not call the API from anywhere but the main thread
- ✓ I shall call the API only, not directly the Revit’s UI
- ✓ I shall use the Idling event cautiously and considerably

Questions?

Autodesk

Autodesk®